109550050 王承勳

Homework 3: Multi-Agent Search

Part I. Implementation (5%)

Part1.

```
# Begin your code (Part 1)
'''
In maxlevel, we try to select the max value from the next level(minlevel).
We first find all the possible actions we can do at this state.
If we arrive the depth we hope or get win or lose situation, we evaluate its value
by evalation function.
Otherwise, we go into the next levels and finally select the max one.
In minlevel, we do the same thing as maxlevel, but this time we choose the min one.

To get the best action, we don't do the first maxlevel with function to get all the
min values generated by the minlevel. Then, we choose all the values that equal the
maxvalue, and choose one from them randomly.
'''

try:
    def maxlevel(gameState, depth):
        cur_depth = depth + 1
        if gameState.isWin() or gameState.isLose() or cur_depth == self.depth:
            return self.evaluationFunction(gameState)
        maxvalue = -float('inf')
        actions = gameState.getLegalActions(0)
        for action in actions:
            nextState = gameState.getNextState(0, action)
            maxvalue = max(maxvalue, minlevel(nextState, cur_depth, 1))
        return maxvalue

    def minlevel(gameState, depth, agentIndex):
        if gameState.isWin() or gameState.isLose():
            return self.evaluationFunction(gameState)
        minvalue = float('inf')
        actions = gameState.getLegalActions(agentIndex)
        for action in actions:
            nextState = gameState.getNextState(agentIndex, action)
            if agentIndex == gameState.getNumAgents()-1:
                minvalue = min(minvalue, maxlevel(nextState, depth))
            else:
                minvalue = min(minvalue, minlevel(nextState, depth, agentIndex+1))
        return minvalue

    actions = gameState.getLegalActions(0)
    scores = []
    for action in actions:
        NextState = gameState.getNextState(0, action)
        score = minlevel(NextState, 0, 1)
        scores.append(score)

    highscore = max(scores)
    bestIndices = [index for index in range(len(scores)) if scores[index] == highscore]
    chosenIndex = random.choice(bestIndices)

    return actions[chosenIndex]
except:
    raise NotImplementedError("To be implemented")
# End your code (Part 1)
```

Part2.

```
# Begin your code (Part 2)
'''
This part does almost the same thing as part1.
However, we use alpha and beta the record the maxvalue and minvalue that
has gotten from other branches on the above level.
Therefore, we take alpha as an example, if we find a value less than alpha
in minlevel, than we can't select it because we have a better choice alpha.
In the situation, we can prune the branch to reduce computation.
'''
try:
    def maxlevel(gameState, depth, alpha, beta):
        cur_depth = depth + 1
        if gameState.isWin() or gameState.isLose() or cur_depth == self.depth:
            return self.evaluationFunction(gameState)
        actions = gameState.getLegalActions(0)
        alpha1 = alpha
        maxvalue = -float('inf')
        for action in actions:
            NextState = gameState.getNextState(0, action)
            maxvalue = max(maxvalue, minlevel(NextState, cur_depth, alpha1, beta, 1))
            if maxvalue > beta:
                return maxvalue
            alpha1 = max(alpha1, maxvalue)
        return maxvalue

    def minlevel(gameState, depth, alpha, beta, agentIndex):
        if gameState.isWin() or gameState.isLose():
            return self.evaluationFunction(gameState)
        actions = gameState.getLegalActions(agentIndex)
        beta1 = beta
        minvalue = float('inf')
        for action in actions:
            NextState = gameState.getNextState(agentIndex, action)
            if agentIndex == gameState.getNumAgents()-1:
                minvalue = min(minvalue, maxlevel(NextState, depth, alpha, beta1))
            else:
                minvalue = min(minvalue, minlevel(NextState, depth, alpha, beta1, agentIndex+
            if minvalue < alpha:
                return minvalue
            beta1 = min(beta1, minvalue)
        return minvalue

    actions = gameState.getLegalActions(0)
    alpha = -float('inf')
    beta = float('inf')
    scores = []
    for action in actions:
        NextState = gameState.getNextState(0, action)
        score = minlevel(NextState, 0, alpha, beta, 1)
        alpha = max(alpha, score)
        scores.append(score)

    highscore = max(scores)
    bestIndices = [index for index in range(len(scores)) if scores[index] == highscore]
    chosenIndex = random.choice(bestIndices)

    return actions[chosenIndex]

except:
    raise NotImplementedError("To be implemented")
# End your code (Part 2)
```

Part3.

```
# Begin your code (Part 3)
'''
The above methods are assuming the adversaries always choose the best actions.
However, it may always contrast to the real world.
Therefore, we deal with this situation by using expect value to replace the
minvalue which can be closer to the reality situation.
We calculate the expect value of all the outcomes from the possible actions
in explevel.
'''
try:
    def maxlevel(gameState, depth):
        cur_depth = depth + 1
        if gameState.isWin() or gameState.isLose() or cur_depth == self.depth:
            return self.evaluationFunction(gameState)
        actions = gameState.getLegalActions(0)
        maxvalue = -float('inf')
        for action in actions:
            NextState = gameState.getNextState(0, action)
            maxvalue = max(maxvalue, explevel(NextState, cur_depth, 1))

        return maxvalue

    def explevel(gameState, depth, agentIndex):
        if gameState.isWin() or gameState.isLose():
            return self.evaluationFunction(gameState)
        actions = gameState.getLegalActions(agentIndex)
        expvalue = 0
        for action in actions:
            NextState = gameState.getNextState(agentIndex, action)
            if agentIndex == gameState.getNumAgents()-1:
                expvalue += maxlevel(NextState, depth)
            else:
                expvalue += explevel(NextState, depth, agentIndex+1)

    actions = gameState.getLegalActions(0)
    scores = []

    for action in actions:
        NextState = gameState.getNextState(0, action)
        score = explevel(NextState, 0, 1)
        scores.append(score)

    highscore = max(scores)
    bestIndices = [index for index in range(len(scores)) if scores[index] == highscore]
    chosenIndex = random.choice(bestIndices)

    return actions[chosenIndex]

except:
    raise NotImplementedError("To be implemented")
# End your code (Part 3)
```

Part4.

```
# Begin your code (Part 4)
'''
This part I will explain in the report.
'''
try:
    newPos = currentGameState.getPacmanPosition()
    NewFood = currentGameState.getFood()
    NewCapsule = currentGameState.getCapsules()
    newGhostStates = currentGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
    newFoodsDistances = [manhattanDistance(newPos, food) for food in NewFood.asList()]
    ScaredGhostStates = [ghostState for ghostState in newGhostStates if ghostState.scaredTimer > 0]
    UnscaredGhostStates = [ghostState for ghostState in newGhostStates if ghostState.scaredTimer == 0]

    score = 0
    minCapsuleDistance = 0
    totalFood = currentGameState.getNumFood()
    currentFood = 1 if not newFoodsDistances else len(newFoodsDistances)
    minScaredGhostDistance = 0 if not ScaredGhostStates \
    else min([manhattanDistance(newPos, state.getPosition()) for state in ScaredGhostStates])

    minGhostDistance = 0 if not UnscaredGhostStates \
    else min([manhattanDistance(newPos, state.getPosition()) for state in UnscaredGhostStates])

    minCapsuleDistance = 0 if not NewCapsule else min([manhattanDistance(newPos, state)for state in NewCapsule])
    newNearestFoodDistance = 0 if not newFoodsDistances else min(newFoodsDistances)
    decisions = len(currentGameState.getLegalActions(0))
    sumScaredTime = sum(newScaredTimes)
    if sumScaredTime > 0:
        if sumScaredTime > minScaredGhostDistance:
            score += (-5)*minScaredGhostDistance + currentGameState.getScore() + 100*len(NewCapsule)
        else:
            score += currentGameState.getScore() + 0.5*minGhostDistance + (-1)*minCapsuleDistance
            score += (-1)*newNearestFoodDistance*weight
    else:

        score += currentGameState.getScore() + 0.5*minGhostDistance
        score += (-1)*newNearestFoodDistance*weight + (-100)*len(NewCapsule) + (-1)*minCapsuleDistance

    return score

except:
    raise NotImplementedError("To be implemented")
# End your code (Part 4)
```

Implementation:

I separate the situations into two parts. The one is that there are some ghosts scared. In this situation, we need to hunt ghosts to get scores. Therefore, I evaluate the states with the distance between the nearest ghost and pacman. However, to avoid chasing the ghosts that are too far to hunt, we act as normal when pacman thinks it's too hard to eat ghosts. The other one is the normal situation, we hope we hope we can always stay in a safer place, being far away the ghost and having many decisions in case that pacman has nowhere to escape. Besides, we still hope we can eat ghosts to get higher scores, so we add significant weights for Capsules. That way, pacman will prefer to eat Capsules but food first. The last, to avoid that pacman stuck in a situation that there is no goal in the limited depths. We give a goal for pacman that it always prefer the place that is closer to the food, and if there is little food, pacman will want to end the game as fast as possible.

Part II. Results & Analysis (5%):

Part1.

```
Question part1
==============

*** PASS: test_cases/part1/0-eval-function-lose-states-1.test
*** PASS: test_cases/part1/0-eval-function-lose-states-2.test
*** PASS: test_cases/part1/0-eval-function-win-states-1.test
*** PASS: test_cases/part1/0-eval-function-win-states-2.test
*** PASS: test_cases/part1/0-lecture-6-tree.test
*** PASS: test_cases/part1/0-small-tree.test
*** PASS: test_cases/part1/1-1-minmax.test
*** PASS: test_cases/part1/1-2-minmax.test
*** PASS: test_cases/part1/1-3-minmax.test
*** PASS: test_cases/part1/1-4-minmax.test
*** PASS: test_cases/part1/1-5-minmax.test
*** PASS: test_cases/part1/1-6-minmax.test
*** PASS: test_cases/part1/1-7-minmax.test
*** PASS: test_cases/part1/1-8-minmax.test
*** PASS: test_cases/part1/2-1a-vary-depth.test
*** PASS: test_cases/part1/2-1b-vary-depth.test
*** PASS: test_cases/part1/2-2a-vary-depth.test
*** PASS: test_cases/part1/2-2b-vary-depth.test
*** PASS: test_cases/part1/2-3a-vary-depth.test
*** PASS: test_cases/part1/2-3b-vary-depth.test
*** PASS: test_cases/part1/2-4a-vary-depth.test
*** PASS: test_cases/part1/2-4b-vary-depth.test
*** PASS: test_cases/part1/2-one-ghost-3level.test
*** PASS: test_cases/part1/3-one-ghost-4level.test
*** PASS: test_cases/part1/4-two-ghosts-3level.test
*** PASS: test_cases/part1/5-two-ghosts-4level.test
*** PASS: test_cases/part1/6-tied-root.test
*** PASS: test_cases/part1/7-1a-check-depth-one-ghost.test
*** PASS: test_cases/part1/7-1b-check-depth-one-ghost.test
*** PASS: test_cases/part1/7-1c-check-depth-one-ghost.test
*** PASS: test_cases/part1/7-2a-check-depth-two-ghosts.test
*** PASS: test_cases/part1/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/part1/7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/part1/8-pacman-game.test

### Question part1: 20/20 ###
```

Part2.

```
Question part2
==============

*** PASS: test_cases/part2/0-eval-function-lose-states-1.test
*** PASS: test_cases/part2/0-eval-function-lose-states-2.test
*** PASS: test_cases/part2/0-eval-function-win-states-1.test
*** PASS: test_cases/part2/0-eval-function-win-states-2.test
*** PASS: test_cases/part2/0-lecture-6-tree.test
*** PASS: test_cases/part2/0-small-tree.test
*** PASS: test_cases/part2/1-1-minmax.test
*** PASS: test_cases/part2/1-2-minmax.test
*** PASS: test_cases/part2/1-3-minmax.test
*** PASS: test_cases/part2/1-4-minmax.test
*** PASS: test_cases/part2/1-5-minmax.test
*** PASS: test_cases/part2/1-6-minmax.test
*** PASS: test_cases/part2/1-7-minmax.test
*** PASS: test_cases/part2/1-8-minmax.test
*** PASS: test_cases/part2/2-1a-vary-depth.test
*** PASS: test_cases/part2/2-1b-vary-depth.test
*** PASS: test_cases/part2/2-2a-vary-depth.test
*** PASS: test_cases/part2/2-2b-vary-depth.test
*** PASS: test_cases/part2/2-3a-vary-depth.test
*** PASS: test_cases/part2/2-3b-vary-depth.test
*** PASS: test_cases/part2/2-4a-vary-depth.test
*** PASS: test_cases/part2/2-4b-vary-depth.test
*** PASS: test_cases/part2/2-one-ghost-3level.test
*** PASS: test_cases/part2/3-one-ghost-4level.test
*** PASS: test_cases/part2/4-two-ghosts-3level.test
*** PASS: test_cases/part2/5-two-ghosts-4level.test
*** PASS: test_cases/part2/6-tied-root.test
*** PASS: test_cases/part2/7-1a-check-depth-one-ghost.test
*** PASS: test_cases/part2/7-1b-check-depth-one-ghost.test
*** PASS: test_cases/part2/7-1c-check-depth-one-ghost.test
*** PASS: test_cases/part2/7-2a-check-depth-two-ghosts.test
*** PASS: test_cases/part2/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/part2/7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/part2/8-pacman-game.test

### Question part2: 25/25 ###
```

Part3.

```
Question part3
==============

*** PASS: test_cases/part3/0-eval-function-lose-states-1.test
*** PASS: test_cases/part3/0-eval-function-lose-states-2.test
*** PASS: test_cases/part3/0-eval-function-win-states-1.test
*** PASS: test_cases/part3/0-eval-function-win-states-2.test
*** PASS: test_cases/part3/0-expectimax1.test
*** PASS: test_cases/part3/1-expectimax2.test
*** PASS: test_cases/part3/2-one-ghost-3level.test
*** PASS: test_cases/part3/3-one-ghost-4level.test
*** PASS: test_cases/part3/4-two-ghosts-3level.test
*** PASS: test_cases/part3/5-two-ghosts-4level.test
*** PASS: test_cases/part3/6-1a-check-depth-one-ghost.test
*** PASS: test_cases/part3/6-1b-check-depth-one-ghost.test
*** PASS: test_cases/part3/6-1c-check-depth-one-ghost.test
*** PASS: test_cases/part3/6-2a-check-depth-two-ghosts.test
*** PASS: test_cases/part3/6-2b-check-depth-two-ghosts.test
*** PASS: test_cases/part3/6-2c-check-depth-two-ghosts.test
*** Running ExpectimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running ExpectimaxAgent on smallClassic after 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/part3/7-pacman-game.test

### Question part3: 25/25 ###
```

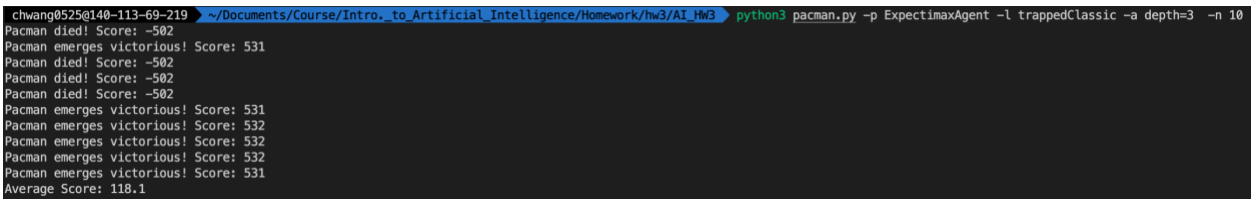
Part4.

```
Question part4
==============

Pacman emerges victorious! Score: 1167
Pacman emerges victorious! Score: 1222
Pacman emerges victorious! Score: 1222
Pacman emerges victorious! Score: 1281
Pacman emerges victorious! Score: 943
Pacman emerges victorious! Score: 1267
Pacman emerges victorious! Score: 1034
Pacman emerges victorious! Score: 1069
Pacman emerges victorious! Score: 1177
Pacman emerges victorious! Score: 1215
Average Score: 1159.7
Scores:        1167.0, 1222.0, 1222.0, 1281.0, 943.0, 1267.0, 1034.0, 1069.0, 1177.0, 1
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/part4/grade-agent.test (8 of 8 points)
*** EXTRA CREDIT: 2 points
***     1159.7 average score (4 of 4 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  2 points
***         >= 1000:  4 points
***     10 games not timed out (2 of 2 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 5:  1 points
***         >= 10:  2 points
***     10 wins (4 of 4 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 4:  2 points
***         >= 7:  3 points
***         >= 10:  4 points

### Question part4: 10/10 ###


Finished at 19:41:26

Provisional grades
==================
Question part1: 20/20
Question part2: 25/25
Question part3: 25/25
Question part4: 10/10
------------------
Total: 80/80
```

Analysis

First, we discuss the minimax method. In my opinion, it performs acceptable. However, like the situation in the trappedClassic, it may perform badly because it supposes the opponents always choose the best way. In the depth = 3, minimax knows that the pacman will die if the adversaries are smart enough, so it choose to collide the ghost to get higher scores for scores decreaing with time. We can find if we decrease the depth to 2. The minimax doesn't know it will die under the worst situation, it may have better performance for the randomness of ghosts' moves. However, in the worst situation, the minimax indeed does the best choice to collide the right ghost, which can get one more score than colliding the left ghost.

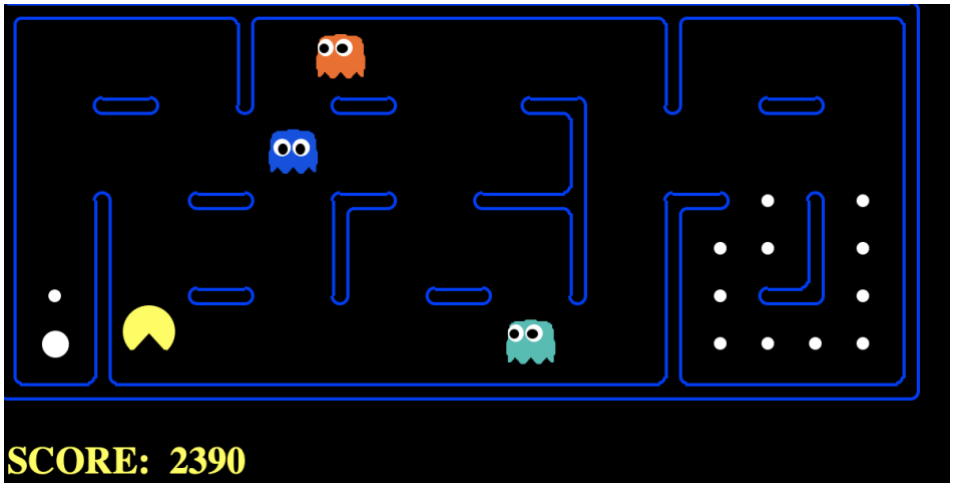
As for expectimax method, it improves the problem that minimax always assumes the worst situation. It calculates the expect value to find which state has the best expect scores. This way can do a decent and a stable decision. In the trappedClassic, we can find that it chooses to go left at first because it supposes the ghost has 50 percents probabilities to go down, not chasing it. If the ghost doesn't chase the pacman, it will get high scores instead of get one more score. Therefore, it chooses to try it.



The next, let's talk about the evaluation function. This really costs me lots of time to try for parameters and coefficients that I should use to get a better performance. The implementation I have mentioned above, so I just summarize some my observations during the trials. The first thing is the depth. I find when the depth = 2, the pacman can't eat the capsules by approaching to decrease the distances. When the pacman just one step from the Capsule, the distances fail to guide it to eat the capsule because there are no way for two steps to get more closer to the Capsule. Besides, the Capsules are always at some dangerous places. Therefore, the pacman just stays there and wait for a stupid ghost passing and it can eat it in two steps. There is another interesting situation about this. When we set the distances as a very great weight, the pacman will be around

the food and Capsules but not eat them because the distances have more to say than the scores.Therefore the pacman just shifts around the states that away from the goal one step.

The last is that Manhattan distances has its limits while using it to find the goal. When our goal is away from the pacman of a wall, it will be like a magnet that be connected to the wall.
In the following picture, the pacman very wants to eat the Capsule, so it tries to get closer to the Capsule. This makes the pacman be captured there.
Update: I change to use the number of capsules to eat the capsules. However, it's 23:55 right now, I don't have time to update some part of my report. I am sorry.



This homework is interesting for trying the ways to evaluate and see what the pacman acts with our evaluations although sometimes it's a…a little bit stupid.